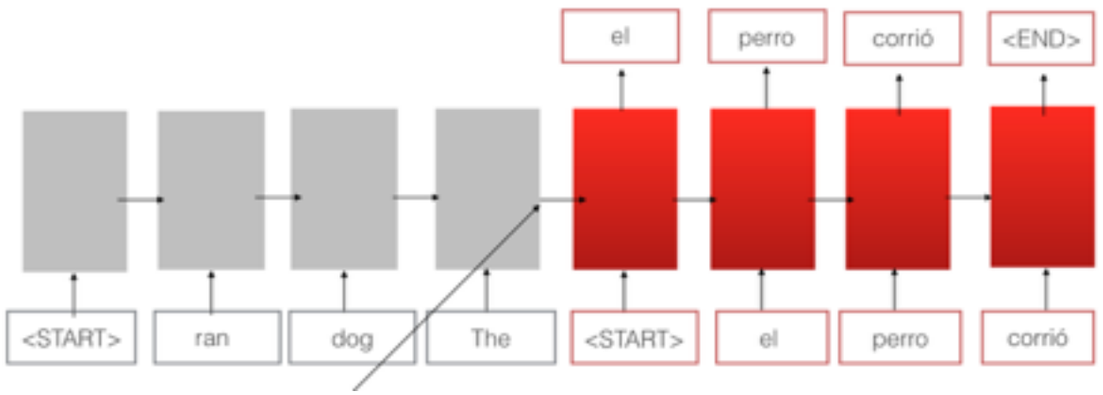
GPU and CPU Recurrent Neural Networks

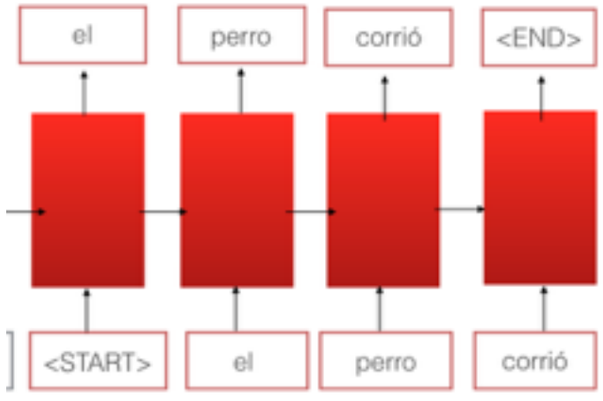
Barret Zoph

Brief RNN Overview

- The RNN architectures in my toolkit have two different types of models
  - Learn functions mapping sequences to sequences (like machine translation)



  - Learn functions learning single sequences (like language modeling)



Program Usage

- If you are ever confused run ./RNN_{GPU,CPU} -h for the help menu! It will show you all the flags and stuff!

- There are 4 things main things you can do with the code:

    1. Train a model (either sequence like LM, or sequence to sequence like MT)

    2. Force-decode (get the per example probability plus perplexity of some dataset, either sequence or sequence to sequence)

    3. Get the k-best paths using beam search for an already trained sequence to sequence model (note the code does not have this for the normal sequence model, as I am not sure why you would want this …)

    4. Stochastic-generation (let your trained sequence model run rampant and generate whatever it desires to. This is what Karpathy did in his blog.)

Training Input Format

- What is the format for training a model using my code? (I will use the GPU version for concreteness, but the syntax is the same of both)

- Note that my code defaults to the sequence to sequence model, so just use the -s flag to train an only sequence model

- The <output file name> is the file where the trained neural network will be stored. It contains all of your model parameters, mappings for integerization (done by me) and the model parameters. This is the only thing you need to keep around once you trained a model!

- The input files must have the tokens separated by spaces. This is the only requirement.

- %RNN_GPU -t <source file name> <target file name> <output file name> /*For sequence to sequence model*/

    - RNN_GPU -t english.txt spanish.txt model.nn /*trains a MT system learning P(spanish | english) */

- %RNN_GPU -s -t <target file name> <output file name> /*for sequence model*/

    - RNN_GPU -s -t english.txt model.nn /*trains a LM learning P(english)*/

Force-Decode Format

- All you need is your neural network file after training! (model.nn in the previous examples)

- %RNN_GPU -f <source file> <target file> <neural network file> <output file> /*for sequence to sequence model*/

    - %RNN_GPU -f english_test.txt spanish_test.txt model.nn output.txt /*get P(spanish_test.txt | english_test.txt)*/

- %RNN_GPU -s -f <target file> <neural network file> <output file> /*for sequence model*/

    - %RNN_GPU -s -f english_test.txt model.nn output.txt /*get P(english_test.txt)*/

kbest path format

- All you need is your neural network file after training! (model.nn in the previous examples)

- kbest does not exist for the sequence model

- %RNN_GPU -k <how many paths> <source file> <neural network file> <output file> /*for sequence to sequence model*/

- %RNN_GPU -k 10 english_test.txt model.nn output.txt /*get the 10 best paths using beam decoding with the input file being english_test.txt*/

Stoch-gen format

- All you need is your neural network file after training! (model.nn in the previous examples)
- This does not exist for the sequence to sequence model

- %RNN_GPU -s -g <neural network file> <output file> /*for sequence model*/

  - %RNN_GPU -s -g model.nn output.txt /*will generate 1000 random tokens to output.txt*/

  - %RNN_GPU —stoch-gen-len 10000 -s -g model.nn output.txt /*will generate 10000 random tokens to output.txt*/

Tips for training

- Do not make your learning rate too high! This can cause your loss function to start increasing. My code starts it at 0.7, as I found this works well for large datasets, but a value of 0.1-0.5 is standard.
- Conversely making your learning rate too small will cause your training to take a very long time to converge
- I have three flags in my code that can help deal with the learning rate.
- 1. (-l,—learning-rate), this will set the starting learning rate
    - RNN_GPU -s -t english.txt model.nn -l 0.3 /*trains a model with initial learning rate of 0.3*/
- 2. (--halve-learning) This will get the perplexity of a user supplied dev set every half epoch and if the perplexity on the dev set increased last half epoch, I reduce the learning rate by an amount specified by the user
    - RNN_GPU -t english.txt spanish.txt model.nn —adaptive-halve-lr english_dev.txt spanish_dev.txt /*trains a model with the english_dev.txt and spanish_dev.txt being used to monitor learning rate halving*/
    - RNN_GPU -s -t english.txt model.nn —adaptive-halve-lr english_dev.txt /*trains a model with the english_dev.txt being used to monitor learning rate halving*/
- 3. (—fixed-halve-learning) This will halve the learning rate every half epoch after a certain epoch has been reached.
    - RNN_GPU -t english.txt spanish.txt model.nn —fixed-halve-lr 5 /*trains a model with the where the learning rate will be halved every epoch after the fifth epoch*/
    - RNN_GPU -s -t english.txt model.nn —fixed-halve-lr 10 /*trains a model with the where the learning rate will be halved every epoch after the tenth epoch*/

Tips for Training

- Make the hidden state size large enough! I use 1000 for all the MT experiments.

- I have a flag to set the number of hidden states (-H,—hiddenstate-size)
    - RNN_GPU -s -t english.txt model.nn -H 200 /*trains a model with 200 hidden states*/

- I would recommend making the hiddenstate size 500+ for any non toy examples. But feel free to play around with it!

- If you make the hidden state size too big you make see a message (“GPU memory alloc failed…), this simply means there is not enough space on the GPU for the model, so reduce the hidden state size

Tips for training

- For sequence to sequence models the source vocabulary can be made arbitrarily big without it hurting the computational cost. The only thing to be careful about is running out of RAM on the GPU.

- By default it sets the source and target sizes to the number of unique tokens that appear in the training data when you train your neural network

- I have two flags to set the vocabulary sizes

  - 1. (-v,—source-vocab), this is only relevant for the sequence to sequence model. This sets the source vocal size

  - 2. (-V,—target-vocab), this sets the target vocal size

Tips for Decoding!

- I set the standard beam-size to 12, but you can change this

- The flag (-b,—beam-size) will set it to whatever you want. Note increasing the beam size does decrease the speed

- If you notice that your neural network is outputting short sentences, play with the flag (-p,—penalty). This is by default zero and adds the amount specified to the unnormalized log prob of each word being decoded. The larger the value, the longer the sentences the model will prefer!

Tips for Training

- Be sure to clip the gradients of these models! I have this on by default with a standard value set (5), but you can again play around with this.

- It is highly recommended that you have gradient clipping on as they gradients can get huge which makes the gradient descent be much more volatile

Tips for training

- If your output vocab is too big and you dont want to have the program unk everything, then try the truncated softmax(-T,—truncated-softmax).
- It will ask for a shortlist (these words always get updated) and a sample size (this is the amount of words that get sampled to be updated at each mini batch)
  - RNN_GPU -t english.txt spanish.txt model.nn —truncated-softmax 10000 5000 /*now a softmax over 15000 only*/

Suggestions

- Find any bugs or have any suggestions to make the user input nicer/easier to use? PLEASE email me at zoph@isi.edu or barretzoph@gmail.com

- https://github.com/isi-nlp/Zoph_RNN